

Dossier de Projet

Anaïs Labit



Projet réalisé en 2023 avec Amine Necib et Alexandre Aloesode

TABLE DES MATIÈRES

| | |
|---|----|
| <u>1. Liste des compétences couvertes</u> | 1 |
| <u>2. Résumé du projet</u> | 2 |
| <u>3. Cahier des charges : besoins et spécifications fonctionnelles</u> | 3 |
| <u>4. Environnement de travail</u> | 4 |
| <u>a. Développement web</u> | 4 |
| <u>b. Outils collaboratifs</u> | 4 |
| <u>c. Documentation et recherches</u> | 4 |
| <u>d. Sécurité et test</u> | 4 |
| <u>5. Réponse aux spécifications techniques</u> | 5 |
| <u>a. Réunion client : moyens mis en oeuvre</u> | 5 |
| <u>b. Explications techniques : extraits de code et argumentation</u> | 7 |
| <u>6. Barre de recherche avec auto complétion : une fonctionnalité représentative des compétences à maîtriser</u> | 24 |
| <u>7. Les vulnérabilités connues et les moyens mis en oeuvre pour y pallier</u> | 27 |
| <u>a. Contrôle d'accès défaillants :</u> | 27 |
| <u>b. Défaillances cryptographiques :</u> | 28 |
| <u>c. Injections :</u> | 29 |
| <u>d. Mauvaises configurations de sécurité :</u> | 30 |
| <u>e. Authentification et identification de mauvaise qualité :</u> | 30 |
| <u>f. Composants vulnérables et obsolètes :</u> | 31 |
| <u>g. Manque d'intégrité des données et du logiciel :</u> | 33 |
| <u>h. Les axes d'améliorations</u> | 33 |
| ● <u>Conception non sécurisée (\neq implémentation) :</u> | 33 |
| ● <u>Carence des systèmes de contrôle et de journalisation (logs) :</u> | 34 |
| ● <u>Falsification de requête côté serveur (SSRF) :</u> | 34 |
| <u>8. Description d'une situation ayant nécessité une recherche, à partir d'un site anglophone</u> | 35 |

1. Liste des compétences couvertes

Le projet couvre les compétences énoncées ci-dessous.

Pour l'activité 1, "Développer la partie front-end d'une application web et web mobile en intégrant les recommandations de sécurité":

- Maquetter une application
- Réaliser une interface utilisateur web ou mobile statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Pour l'activité 2, "Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité.":

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

2. Résumé du projet

Ce document présente le processus de création d'une boutique en ligne de vente de paniers de produits bio et locaux, réalisée pour un client fictif.

Le développement s'est articulé autour de plusieurs aspects.

Après validation des fonctionnalités et du design, le front-end a été conçu avec une interface responsive mettant en avant les produits phares. Un système de création de comptes sécurisés a été développé, permettant aux utilisateurs de s'inscrire, gérer leur profil, noter les produits, etc.

Le back-end permet la création de différents types de profils d'utilisateurs, ainsi l'utilisation d'un tableau de bord pour la gestion des produits et des catégories. La sécurité a été assurée en évitant l'exposition des informations sensibles côté client et en vérifiant les autorisations et les rôles côté client et côté serveur.

Pour répondre aux besoins du projet, divers outils et langages ont été utilisés.

Figma a été utilisé pour le maquettage du site. L'éditeur de code principal était Visual Studio Code, et la collaboration en temps réel a été facilitée par Visual Studio Code Live Share. Le versioning a été réalisé avec Git et GitHub, tandis que Trello a été utilisé pour la planification des tâches.

Les langages HTML, CSS, JavaScript et PHP ont été employés pour l'intégration, la validation des formulaires, le traitement des données et les opérations en base de données. La gestion de la base de données s'est faite avec phpMyAdmin, compatible avec MySQL.

Toutes les étapes détaillant la création d'une boutique en ligne fonctionnelle et sécurisée, offrant une expérience utilisateur agréable et fluide, seront détaillées dans les pages qui suivent.

3. Cahier des charges : besoins et spécifications fonctionnelles

Le site devait à minima inclure :

| | |
|------------------|---|
| FRONT-END | <ul style="list-style-type: none">- Page d'accueil attractive avec produits phares- Design contemporain et responsive- Barre de recherche avec auto compléction en JavaScript- Accès à la boutique avec filtres par catégorie/sous-catégorie- Pages de détails des produits générées dynamiquement- Système de création de comptes d'utilisateurs- Module d'inscription/connexion et gestion de profil utilisateur- Tableau de bord administrateur pour la gestion des produits et catégories- Système de validation du panier (simulation de paiement) |
| BACK-END | <ul style="list-style-type: none">- Gestion des produits à l'aide d'un back office : ajout, suppression et modification de produits.- Gestion des catégories et sous-catégories de produits : ajout, suppression et modification.- Système de création de comptes utilisateurs : module d'inscription/connexion avec utilisation de JavaScript et asynchronicité.- Page de gestion du profil utilisateur : récapitulatif, possibilité de modifier les informations, historique d'achat, consultation du panier.- Système de validation du panier : implémentation d'une simulation de validation sans nécessité d'une vraie solution de paiement. |

4. Environnement de travail

a. Développement web

Visual Studio Code comme éditeur de code
HTML et CSS pour l'intégration
JavaScript pour la validation des formulaires et récupérer les informations côté client
PHP pour le traitement des données des formulaires, les contrôles et le CRUD (create, read, update et delete) en base de données
AJAX pour éviter le rechargement de la page
Font Awesome pour l'importation d'icônes gratuites
Google Fonts pour le choix des polices
Media queries pour le responsive design
W3C Validator pour valider le code aux normes du web
phpMyAdmin comme Service de Gestion de Bases de Données Relationnelles compatible avec MySQL
PDO (PHP Data Objects) pour interagir avec la base de données
Plesk pour l'hébergement du site

b. Outils collaboratifs

Git et GitHub pour le versioning
Trello pour la planification des tâches

Figma pour le maquettage
Visual Studio Code Live Share pour collaborer en temps réel sur Visual Studio Code

c. Documentation et recherches

Google
Stackoverflow en anglais
Forums
OpenClassroom en français
MdN web docs en anglais
Documentations officielles : sql.sh et php.net en français et anglais

d. Sécurité et test

synk.io
dependency-check.sh
SQLMap pour tester les vulnérabilités potentielles

5. Réponse aux spécifications techniques

a. Réunion client : moyens mis en oeuvre

“Cher client,

L'analyse approfondie de vos besoins a été réalisée afin de comprendre les exigences spécifiques de ce projet. Sur la base de cette analyse, nous avons créé un [**wireframe et une maquette**](#) détaillés de votre boutique en ligne, reflétant fidèlement vos attentes et vos besoins, tant en termes de fonctionnalités que de design. Cette étape cruciale nous a permis de **planifier et de visualiser** la structure et le design de la boutique en ligne, en organisant les différentes sections et fonctionnalités de manière efficace.

Suite à l'approbation de la maquette, nous avons procédé au développement du site de commerce électronique, en tenant compte des fonctionnalités et des spécifications discutées lors de nos réunions.

La première étape a été la création d'[**une base de données relationnelle**](#), conçue pour stocker toutes les informations nécessaires au fonctionnement du site, assurer une structure logique et cohérente et faciliter l'évolutivité du site au besoin.

Pour le front-end, nous avons conçu une [**page d'accueil attractive**](#) mettant en avant vos produits phares. Le design est contemporain et le site est [**entièrement responsive**](#), s'adaptant parfaitement aux écrans desktop, mobiles et tablettes.

Pour faciliter l'expérience utilisateur, nous avons développé un système de [**création de comptes sécurisés**](#) utilisant JavaScript et l'asynchronicité pour une expérience fluide. Les utilisateurs peuvent s'inscrire et [**se connecter facilement**](#), [**gérer leur profil utilisateur**](#), [**consulter leur historique**](#) d'achat, validé ou non, depuis une page dédiée. Le système de [**validation du panier**](#) a été implémenté avec succès. Bien qu'il s'agisse d'une [**simulation de paiement**](#), il permet aux utilisateurs de passer par toutes les étapes de validation de leur panier, garantissant ainsi une expérience utilisateur complète. Un utilisateur pourra créer un panier sans compte, c'est au moment de valider le panier, qu'il lui sera demandé de s'inscrire.

Nous avons mis en place une **barre de recherche avec auto-complétion** en JavaScript, permettant aux utilisateurs de trouver rapidement les produits qu'ils recherchent. L'accès à la boutique a été optimisé avec des **filtres par catégorie et sous-catégorie**, offrant une navigation facile et intuitive. Nous avons mis en place une **pagination** qui, elle aussi, rend l'expérience de navigation plus agréable et moderne. Les pages de détails des produits sont générées **dynamiquement**, fournissant aux utilisateurs des informations détaillées sur chaque produit. Nous avons intégré un système de **notation des produits**, permettant aux utilisateurs de laisser des commentaires et des avis, favorisant ainsi l'interaction et l'engagement des clients.

En ce qui concerne le back-end, nous avons mis en place plusieurs types de profil : particulier, entreprise, administrateur et enfin, collaborateur. L'administrateur a accès à un **tableau de bord**. Nous avons en effet développé un **back office** permettant la gestion des produits, des catégories et des sous-catégories. Vous pouvez désormais ajouter et modifier facilement les produits et les catégories selon vos besoins, en y associant les images que vous souhaitez. Nous avons utilisé la technologie AJAX afin de récupérer **en temps réel**, et donc de manière **dynamique**, des informations sur les commandes depuis la base de données. Ce tableau de bord vous permet également de gérer les rôles des différents utilisateurs.

Nous avons accordé une grande importance à la **sécurité et à la protection des données**. Le code est conçu de manière à ne pas exposer les informations sensibles côté client, et les autorisations et les rôles des utilisateurs sont vérifiés tant côté client que côté serveur, garantissant ainsi une expérience utilisateur sécurisée. En utilisant une **classe abstraite**, nous avons pu mettre en œuvre des couches de sécurité supplémentaires pour protéger vos données sensibles. Cette dernière agit comme une interface sécurisée entre le client et le serveur, en s'assurant que seules les informations nécessaires sont échangées et que les données confidentielles sont protégées. Par ailleurs, votre site est conforme aux normes de confidentialité et de protection des données, notamment au **RGPD**, assurant ainsi la sécurité des informations personnelles des utilisateurs.

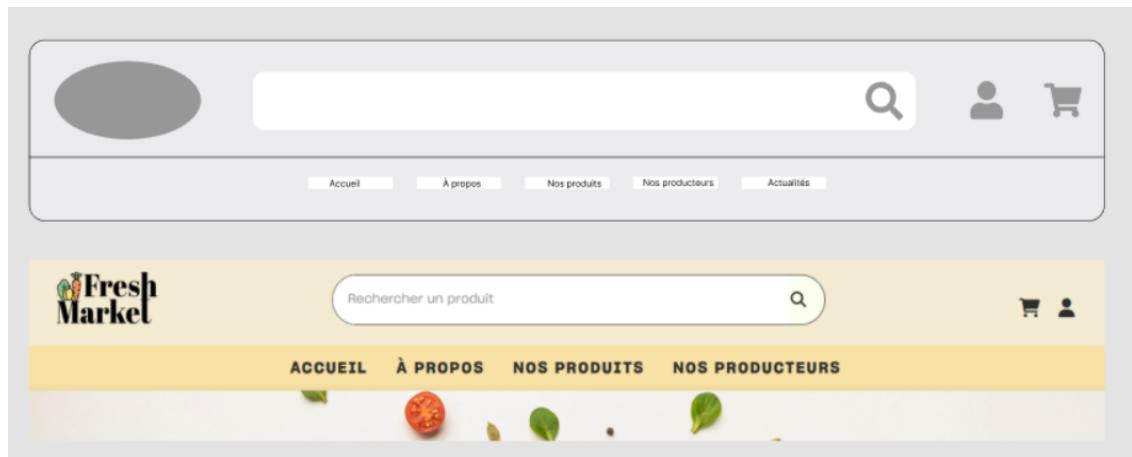
Nous avons également **optimisé le site pour les moteurs de recherche** en mettant en place des techniques de **référencement naturel**, telles que l'utilisation de balises HTML appropriées, la rédaction de descriptions pertinentes et l'adoption d'URL conviviales. Cela permettra d'améliorer la visibilité de votre site et son classement dans les résultats de recherche.

Pour finir, nous avons mis en place une **architecture MVC (Model-View-Controller)** qui favorise la séparation des problèmes et facilite la maintenance du code. L'adoption de cette architecture MVC garantit une meilleure **évolutivité et maintenabilité** du code, ce qui facilitera les futures mises à jour et l'ajout de nouvelles fonctionnalités à votre boutique en ligne.

Cordialement,
L'équipe de développement"

b. Explications techniques : extraits de code et argumentation

- wireframe et maquette



L'analyse des besoins et des spécifications du projet a été effectuée en premier lieu. La création d'un wireframe et d'une maquette détaillée a permis de planifier et de visualiser la structure et le design de la boutique en ligne. Nous avons privilégié la simplicité, la clarté, la cohérence visuelle et l'adaptabilité, tout en tâchant de rendre visibles les différentes interactions et fonctionnalités du site. Cette étape a permis d'organiser le travail de manière efficace.

Cela nous a également permis de créer un style global cohérent pour l'interface utilisateur, que l'on a pu retranscrire facilement dans notre feuille de style.

```
/* ===== IMPORT CUSTOM FONTS ===== */
@font-face {
    font-family: "Sharp Grotesk Semi-Bold 20";
    src: url("./assets/fonts/SharpGrotesk-SemiBold20.otf) format("opentype");
}
@font-face {
    font-family: "Sharp Grotesk Book 20";
    src: url("./assets/fonts/SharpGrotesk-Book20.otf) format("opentype");
}

/* ===== DECLARATION VARIABLES CSS ===== */
:root {
    /* Fonts */
    --title-font: "Sharp Grotesk Semi-Bold 20", sans-serif;
    --body-font: "Sharp Grotesk Book 20", sans-serif;
    /* Font-Sizing */
    --text-xxl: 80px;
    --text-xl: 50px;
    --text-l: 36px;
    --text-m: 26px;
    --text-s: 20px;
    --text-xs: 16px;
    --link-underline: underline;

    /* Colors */
    /* PRIMARY COLORS */
    --black: #000000;
    --light-black: #2d2d2d;
    --grey: #666666;
    --ivory: #fbffed;
    --ivory-transparent: rgba(251, 255, 237, 0.9);
    --white: #ffffff;

    /* SECONDARY COLORS */
    --seagreen: #2e8540;
    --khaki: #b0c472;
    --lin: #ffff9ea;
    --ecru: #f2ebd1;
    --moccasin: #f8e1a4;
    --navajowhite: #f2d79a;
    --ambre: #f1c34a;
    --venitien: #dea455;
    --sienna: #9a4726;

    /* BOX STYLE */
    --border: 0.1rem solid var(--light-black);
    --border-hover: 0.1rem solid var(--black);
    --shadow: 0 0 0.5rem 1rem rgba(0, 0, 0, 0.308);
}
```

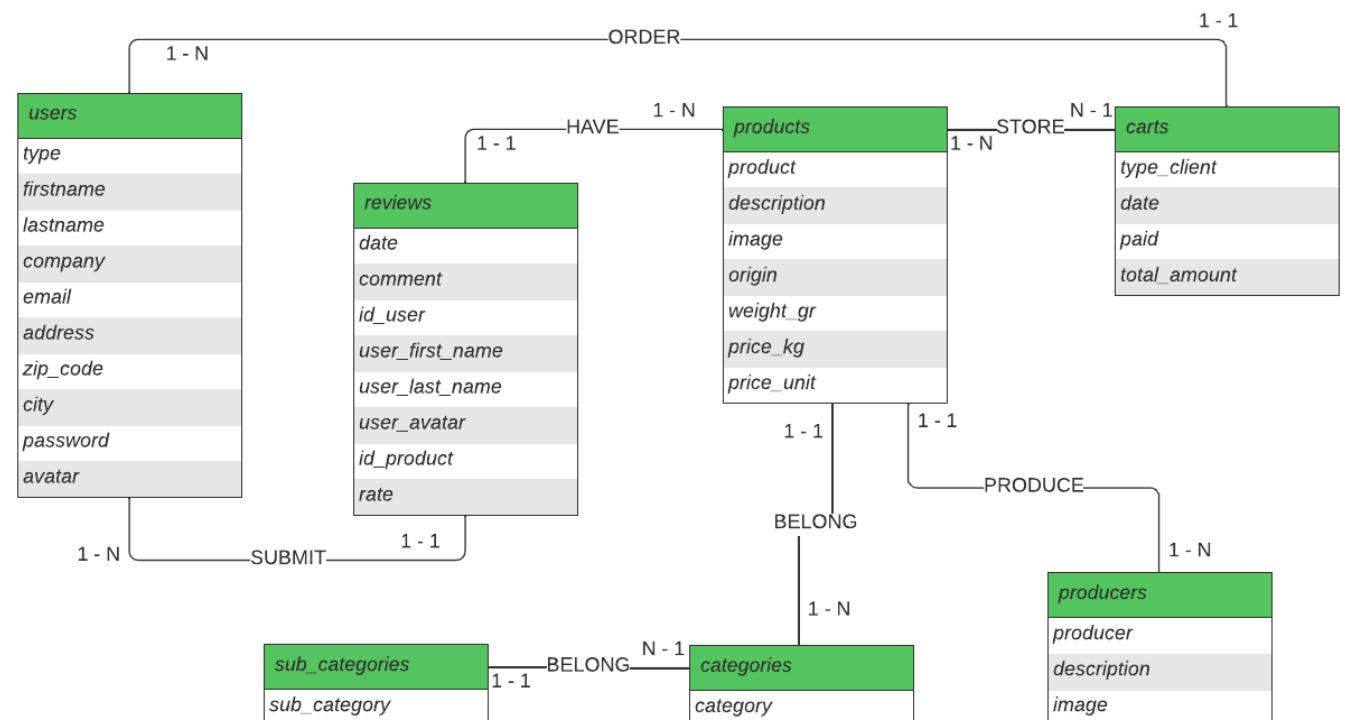
- **base de données relationnelle**

La base de données a été conçue pour stocker toutes les informations nécessaires au fonctionnement du site, telles que les produits, les utilisateurs, les catégories, etc. Une table de liaison a été mise en place pour gérer les relations entre ces différentes entités, assurant ainsi une structure logique et cohérente.

Le MCD, le MPD et le MLD sont des modèles utilisés pour concevoir et organiser la structure des données dans un système d'information. Le MCD fournit une vue globale des entités et des relations, le MPD traduit cette structure en une base de données réelle et le MLD agit comme une représentation intermédiaire indépendante du SGBD. Ensemble, ces modèles facilitent la conception et le déploiement des données dans un site.

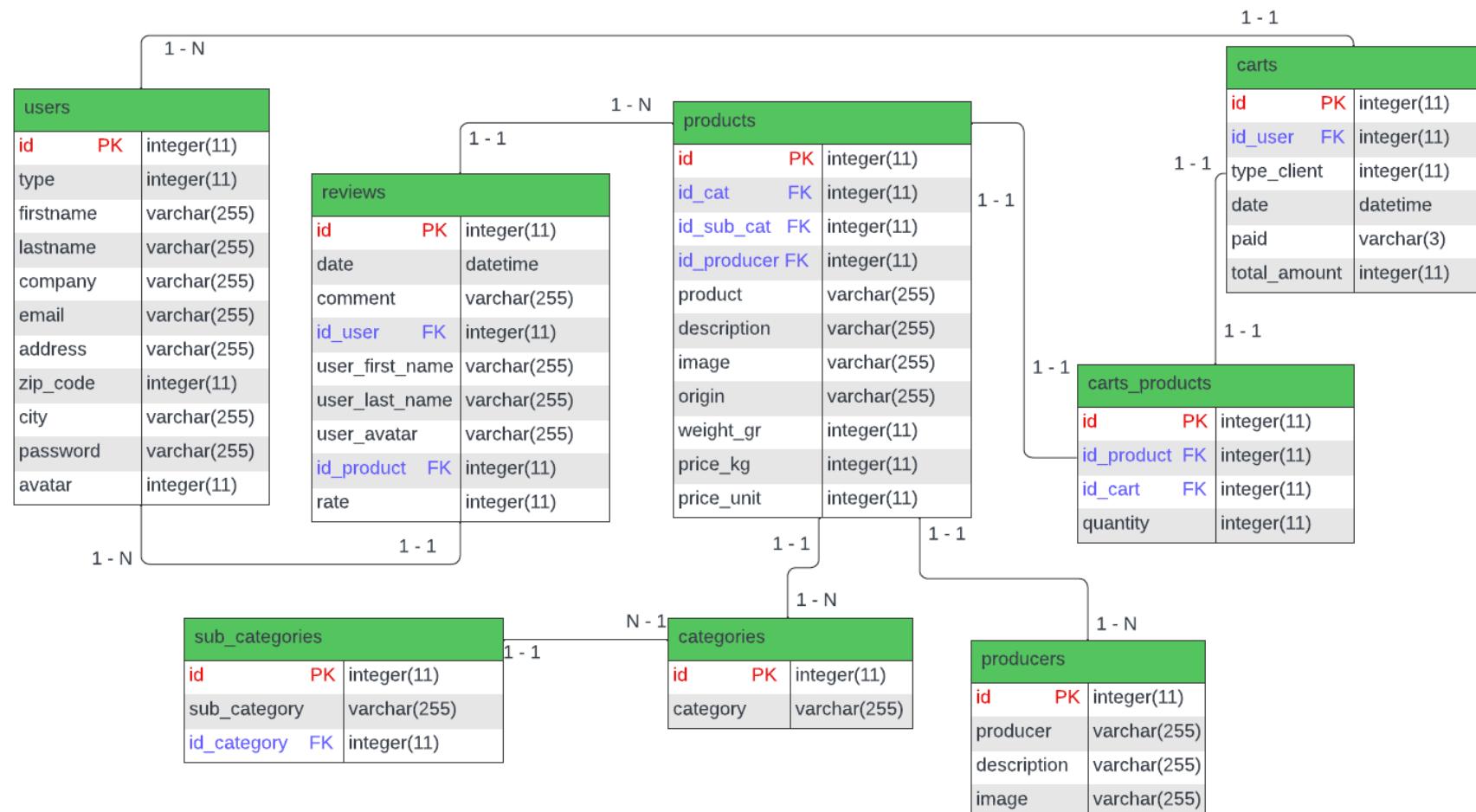
1. Modèle Conceptuel de Données (MCD) :

Le MCD est une représentation visuelle des entités (objets) et des relations entre ces entités. Il permet de définir la structure globale des données, en identifiant les entités principales et leurs attributs, ainsi que les relations qui les lient. Le MCD facilite la conception de la base de données et l'organisation des données de manière cohérente.



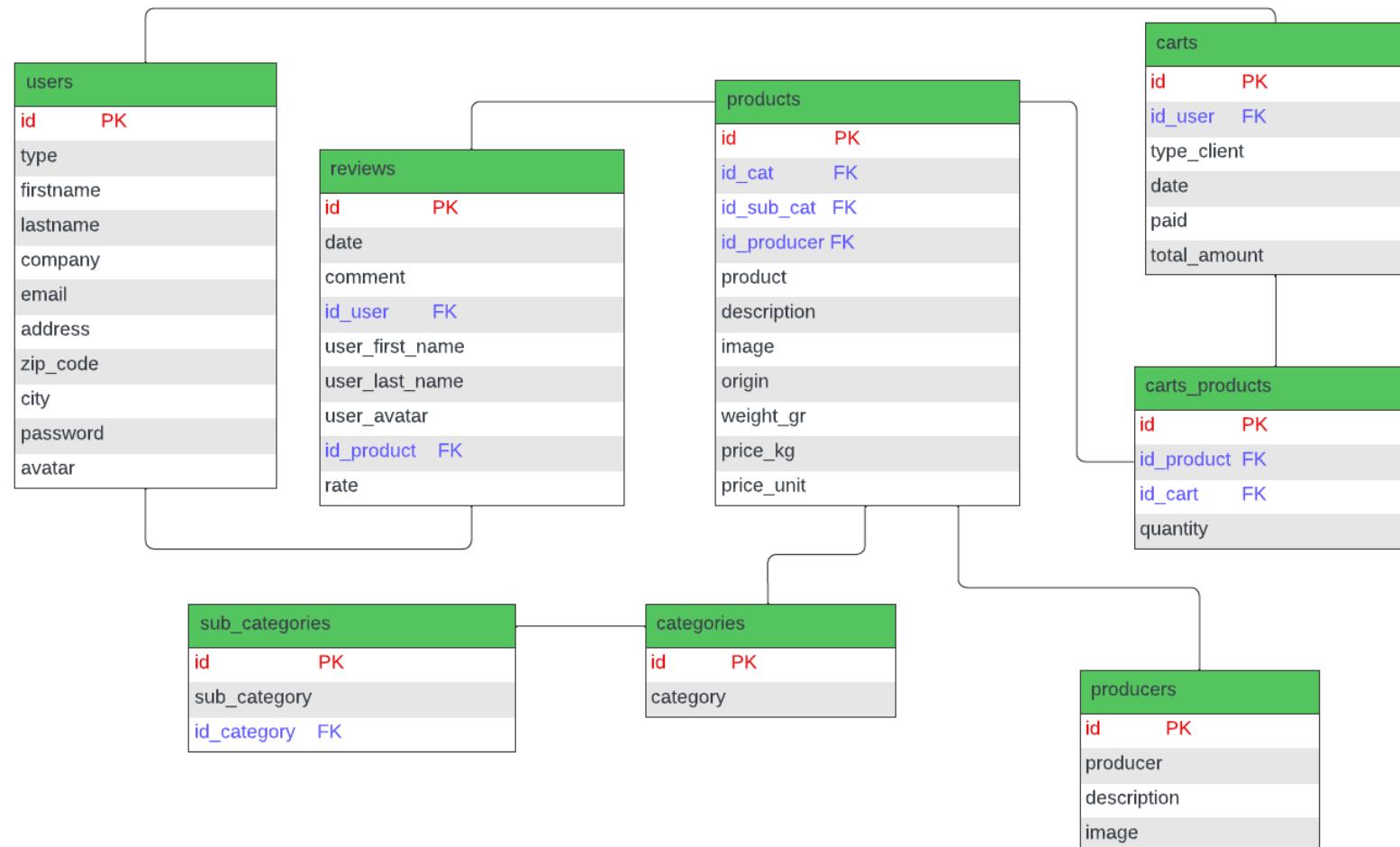
3. Modèle Physique de Données (MPD) :

Le MPD est une représentation concrète des structures de données du MCD dans un langage spécifique à un Système de Gestion de Base de Données (ou "SGBD"). Il définit les tables, les colonnes, les clés primaires et étrangères, les contraintes et les index nécessaires pour stocker et manipuler les données. Le MPD traduit le MCD en une structure de base de données réelle. Il permet de mettre en place une base de données fonctionnelle en définissant les tables et les relations nécessaires pour stocker et gérer les données du site.



2. Modèle Logique de Données (MLD) :

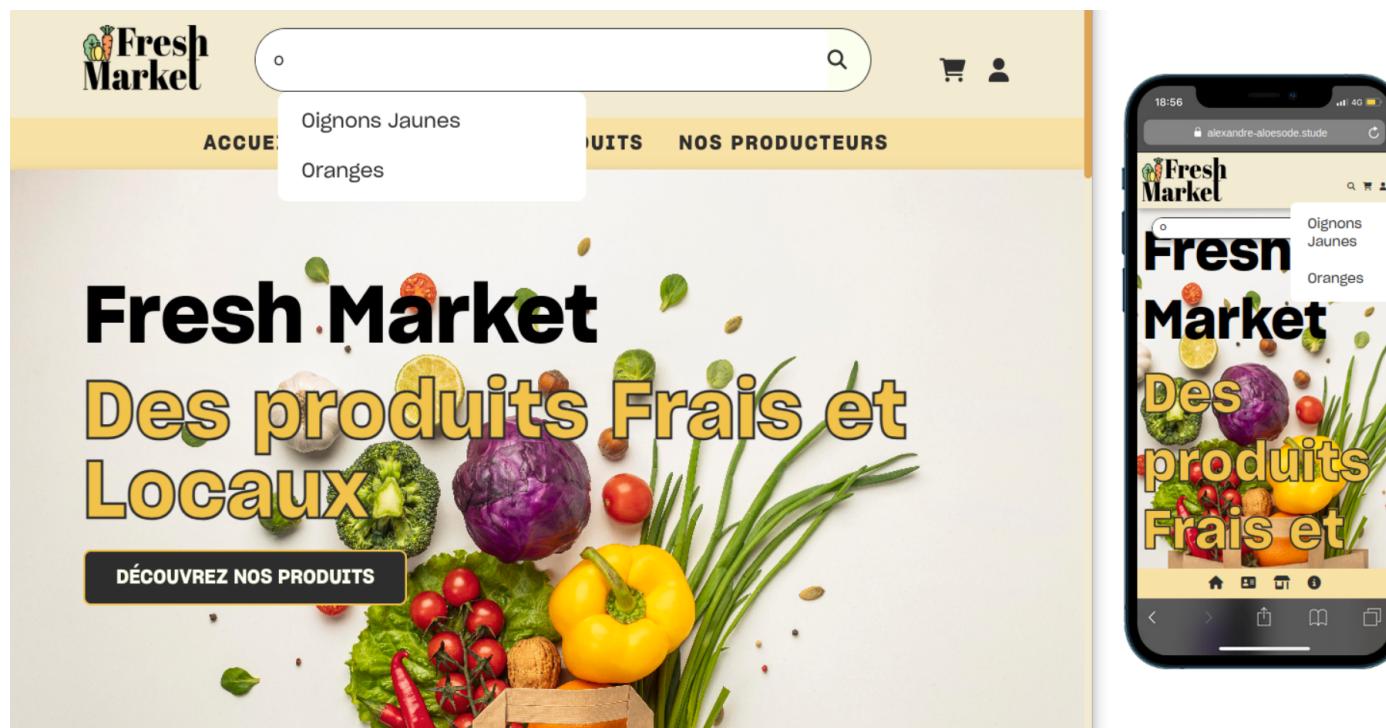
Le MLD est une combinaison du MCD et du MPD. Il fournit une représentation intermédiaire entre le MCD et le MPD et représente la structure des données de manière indépendante d'un SGBD spécifique. Il utilise un langage de modélisation des données, tel que le langage UML (ou "Unified Modeling Language"), pour décrire les entités, les attributs et les relations du système d'information.



- une page d'accueil attractive...

Nous avons mis en place une page d'accueil attrayante qui respecte les principes fondamentaux de l'UI/UX afin d'offrir une expérience agréable et engageante. Nous avons structuré la page d'accueil de manière à ce que les éléments importants tels que le logo, le titre principal et les appels à l'action (ou "Call To Action") soient facilement identifiables : les menus et les liens sont clairement visibles et bien organisés, permettant aux utilisateurs de trouver rapidement les informations recherchées. Nous avons intégré des boutons et des liens qui incitent les visiteurs à effectuer des actions spécifiques, comme s'inscrire ou faire une recherche.

Nous avons sélectionné des visuels attractifs qui contribuent à transmettre efficacement notre message. Nous utilisons des titres accrocheurs, des sous-titres informatifs et des paragraphes courts pour faciliter la lecture, le tout en s'appuyant sur une typographie lisible et adaptée au contenu. Nous avons choisi une palette de couleurs cohérente qui correspond à l'identité de la marque et crée une atmosphère harmonieuse. Enfin, nous avons utilisé des espaces vides, dits "négatifs", de manière stratégique pour permettre à la page de respirer et mettre en valeur les éléments importants.

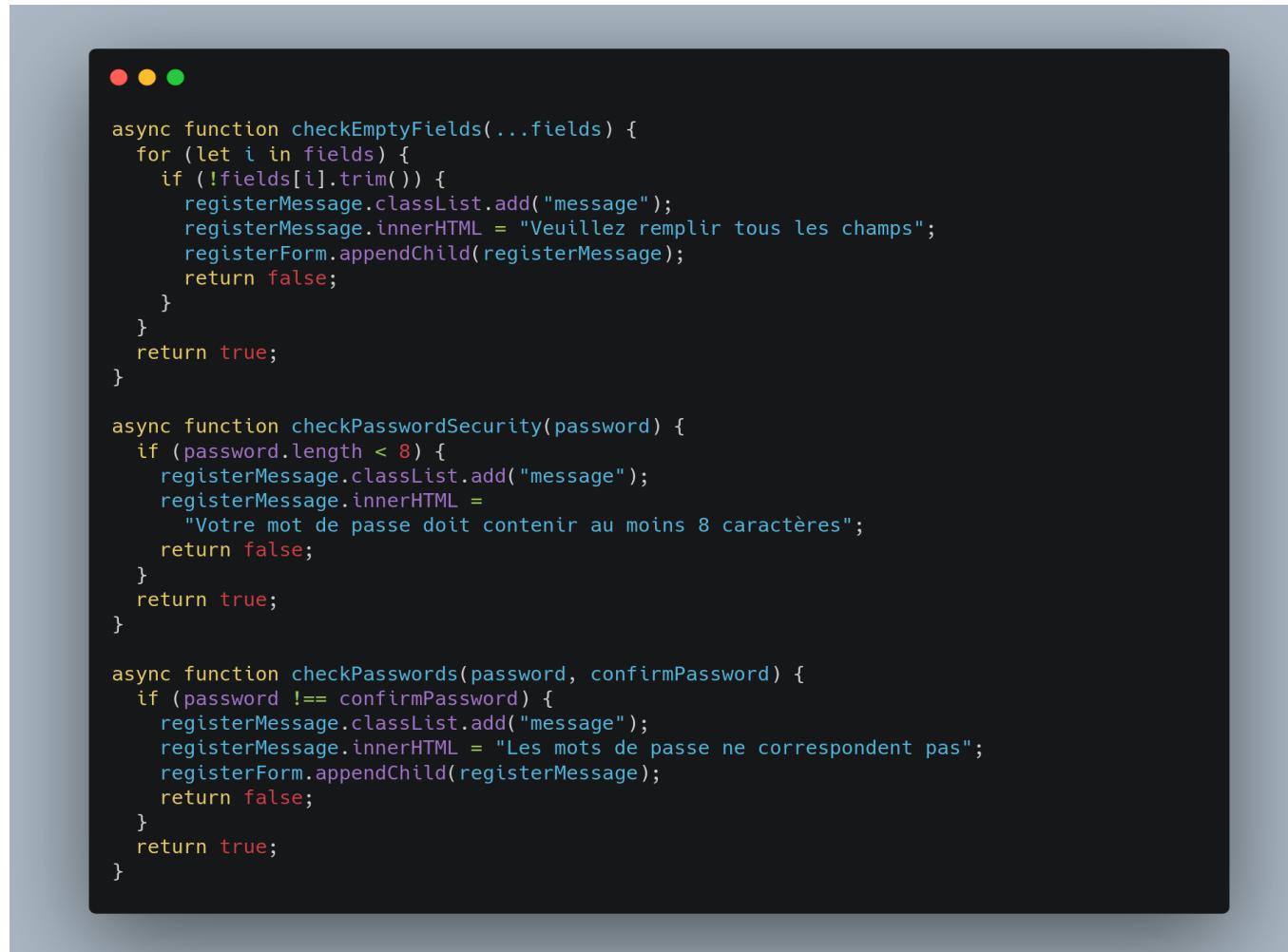


- et entièrement responsive

Un aspect important du projet était d'assurer la responsivité du site, en garantissant son adaptation fluide et esthétique à divers appareils et tailles d'écran. Ainsi, les utilisateurs pouvaient naviguer et effectuer des achats de manière pratique, que ce soit depuis un ordinateur, une tablette ou un smartphone. Cela est notamment passé par la mise en place d'un menu spécifique qui s'adapte à l'espace limité de l'écran et facilite la navigation.

- **création de comptes sécurisés**

Après avoir récupéré la valeur des champs, cette partie du code gère l'enregistrement d'un nouvel utilisateur en vérifiant la sécurité du mot de passe, la correspondance des mots de passe, et en effectuant des vérifications sur les champs vides. Si toutes les conditions sont respectées, alors, l'utilisateur sera effectivement ajouté en base de données.



```
async function checkEmptyFields(...fields) {
  for (let i in fields) {
    if (!fields[i].trim()) {
      registerMessage.classList.add("message");
      registerMessage.innerHTML = "Veuillez remplir tous les champs";
      registerForm.appendChild(registerMessage);
      return false;
    }
  }
  return true;
}

async function checkPasswordSecurity(password) {
  if (password.length < 8) {
    registerMessage.classList.add("message");
    registerMessage.innerHTML =
      "Votre mot de passe doit contenir au moins 8 caractères";
    return false;
  }
  return true;
}

async function checkPasswords(password, confirmPassword) {
  if (password !== confirmPassword) {
    registerMessage.classList.add("message");
    registerMessage.innerHTML = "Les mots de passe ne correspondent pas";
    registerForm.appendChild(registerMessage);
    return false;
  }
  return true;
}
```

- validation du panier et simulation de paiement

```
// AbstractModel.php
    public function updateOne(array $params)
    {
        $fieldsToUpdate = $params;
        array_pop($fieldsToUpdate);
        $requestString = [];
        foreach ($fieldsToUpdate as $key => $value) {
            $fieldName = str_replace(':', ' ', $key);
            $requestString[] = $fieldName . ' = ' . $key;
        }
        $requestString = implode(', ', $requestString);
        $requestUpdateOne = "UPDATE $this->tableName SET $requestString WHERE id = :id";
        $queryUpdateOne = self::getPdo()->prepare($requestUpdateOne);
        $queryUpdateOne->execute($params);
    }

// CartModel.php
    public function validateCart(array $params): void
    {
        $this->tableName = "carts";
        $this->updateOne($params);
    }

// payment.php :
if(isset($_POST['validateCart']))
{
    $cart = new CartModel();
    $cart->validateCart([
        ":paid" => "YES",
        ":id" => $_SESSION['cartId'][0]]);
    echo json_encode(["success" => true , "message" => "Paiement validé"]);
}
```

La validation du panier passe par le changement de statut de ce dernier en base de données. Par défaut, le champ “paid” est “NO”. Lorsque que l’utilisateur procède au renseignement de ses informations de paiement (qui passe par la vérification de leur conformité grâce à des regex), on utilise la méthode `UpdateOne()` héritée de l’`AbstractModel` pour mettre à jour le champ.

1. Dans le fichier “`AbstractModel.php`”, la méthode `updateOne()` est définie. Elle prend un tableau de paramètres en entrée et met à jour les champs d’une table spécifique en utilisant une requête SQL “`UPDATE`”. Les champs à mettre à jour sont extraits du tableau de paramètres fourni. Ensuite, une requête SQL est construite en utilisant les noms des champs et les paramètres correspondants, puis exécutée.

2. Dans le fichier “`CartModel.php`”, la méthode `validateCart()` est définie. Elle prend un tableau de paramètres en entrée et met à jour les champs de la table “`carts`” en utilisant la méthode `updateOne()` héritée de la classe “`AbstractModel`”. La variable `$this->tableName` est définie pour indiquer la table sur laquelle effectuer la mise à jour.

3. Dans le fichier "payment.php", lorsque le formulaire de paiement est soumis, les valeurs des champs de paiement ("paymentCard", "paymentMonth", "paymentCVV") sont récupérées à partir de `$_POST`. Ensuite, une vérification est effectuée pour s'assurer que tous les champs sont remplis. Si des champs sont vides, un message d'erreur est renvoyé en format JSON.

4. Si tous les champs sont remplis, une instance de la classe "CartModel" est créée. La méthode `validateCart()` est appelée en lui passant un tableau de paramètres comprenant la valeur ":paid" définie sur "YES" et l'ID du panier stocké dans la variable de session `$_SESSION['cartId'][0]`. Cela met à jour le champ "paid" du panier correspondant dans la table "carts".

5. Enfin, une réponse JSON est renvoyée indiquant que le paiement a été validé avec succès.

Il est important de noter que toutes les étapes de contrôle ont été exclues du code fourni.

The screenshot shows the Fresh Market website's shopping cart page. At the top, there is a navigation bar with the logo "Fresh Market", a search bar containing "Rechercher un produit", and a shopping cart icon showing "2" items. Below the navigation bar, there are menu links: ACCUEIL, À PROPOS, NOS PRODUITS, and NOS PRODUCTEURS. The main content area is titled "Votre panier". Inside this area, there is a yellow box containing two items: "Haricots Verts" (1 unit, 7€) and "Amandes" (2 units, 5€). Each item has quantity controls (+, -, delete) and a small thumbnail image. Below the yellow box, the total is displayed as "Total : 13.58€". At the bottom of the page, there is a green button labeled "Valider le panier".

- gérer son profil utilisateur

La fonction qui suit utilise la même méthode abstraite que celle présentée pour mettre à jour le statut du panier. Compte tenu de toutes les vérifications qu'elle implique avant l'instanciation de la méthode (`$userModel->updateOne($valuesToSend);`), cette dernière n'est simplement pas re-détaillée.

`UpdateSelf()` permet de mettre à jour le profil de l'utilisateur en envoyant les données du formulaire `updateForm` au serveur via une requête POST. Les données du formulaire sont encapsulées dans un objet `FormData`. Un paramètre supplémentaire "updateProfile" est ajouté au `FormData`. La requête est effectuée en utilisant la fonction `fetch()`, en spécifiant l'URL du fichier "user_management.php" et les options définies. La réponse de la requête est ensuite traitée. Si des erreurs sont renvoyées par le serveur (indiquées par la propriété `errors` de l'objet JSON), elles sont affichées dans l'élément HTML avec l'ID "msgContainer". Sinon, le message de succès (indiqué par la propriété `success` de l'objet JSON) est affiché à la place.

```
async function updateSelf() {
    const reqUpdate = new FormData(updateForm);

    reqUpdate.append("updateProfile", "updateProfile");
    const options = {
        method: "POST",
        body: reqUpdate,
    };

    const updateUser = await fetch("../src/Routes/user_management.php", options);
    const msg = await updateUser.json();

    document.querySelector("#msgContainer");
    if (msg.errors) {
        msgContainer.innerHTML = msg.errors;
    } else {
        msgContainer.innerHTML = msg.success;
    }
}
```

- **filtres par catégories et sous-catégories**

```
//product.js
async function displaySingleCategory() {
    productsDiv.innerHTML = "";
    const singleCategoryForm = new FormData();
    singleCategoryForm.append("displaySingleCategory", "displaySingleCategory");
    singleCategoryForm.append("categoryId", this.value);

    const requestsingleCategoryOptions = {
        method: "POST",
        body: singleCategoryForm
    }

    const fetchSingleCategory = await fetch("../src/Routes/product_display.php", requestsingleCategoryOptions)
    const singleCategory = await fetchSingleCategory.json()
    rawDisplay(singleCategory)
    displaySingleSubCategoriesButtons(this.value);

}

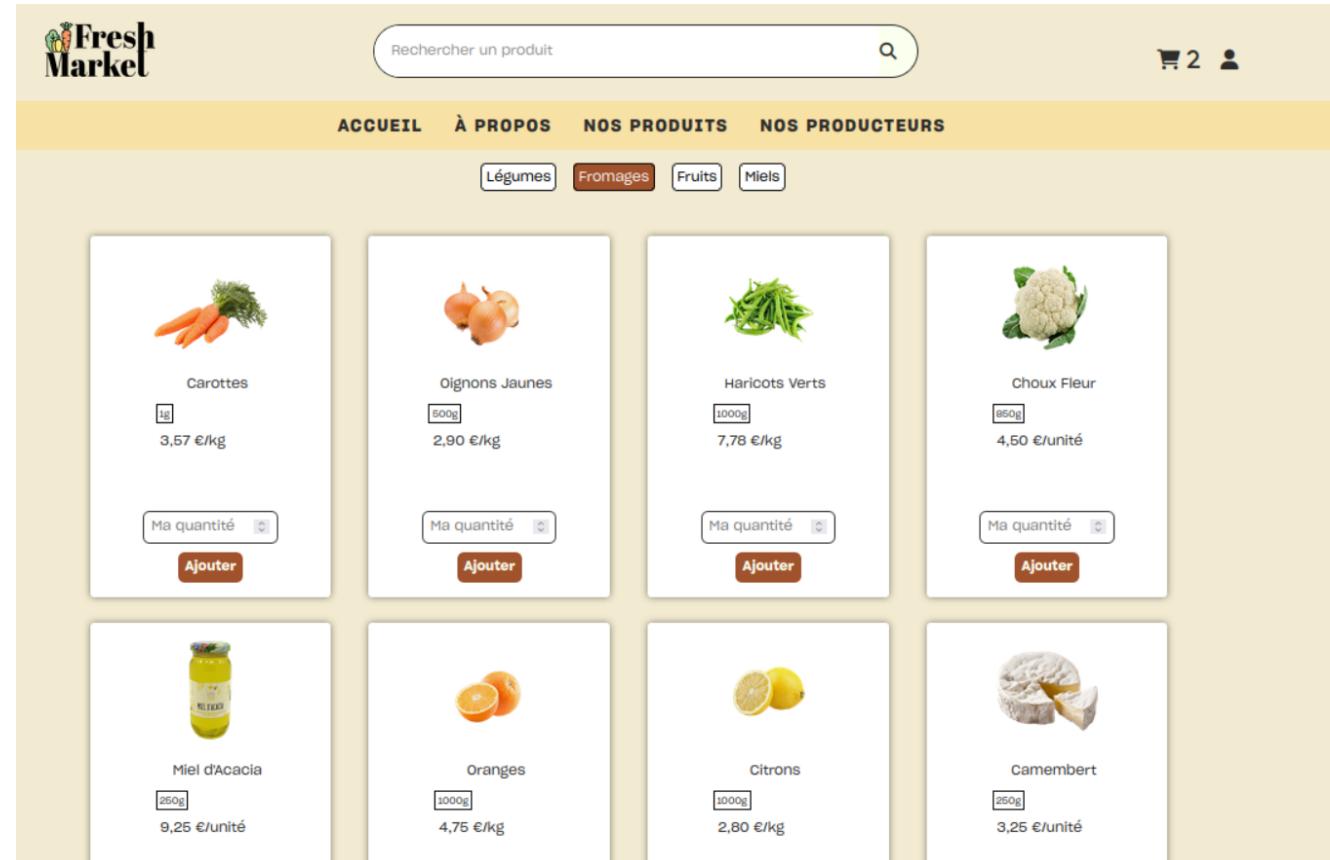
//product_display.php
if(isset($_POST['displaySingleCategory'])) {
    $singleCategoryProducts = new ProductModel();
    $singleCategoryProductsList =
    $singleCategoryProducts->readOnebyForeignKey('id_cat',int)$_POST['categoryId'], "void");
    echo json_encode($singleCategoryProductsList);
}

//AbstractModel.php
public function readOnebyForeignKey(string $foreignkey, int $keyValue, $order): array
{
    $requestReadOne = "SELECT * FROM $this->tableName WHERE $foreignkey = :$foreignkey";

    $queryReadOne = self::getPdo()->prepare($requestReadOne);
    $queryReadOne->execute([':' . $foreignkey => $keyValue]);
    $resultReadOne = $queryReadOne->fetchAll();
    return $resultReadOne;
}
```

Les extraits de code fournis expliquent comment fonctionne le système de filtre de produits :

1. Dans le fichier "product.js", la fonction `displaySingleCategory()` est définie. Elle est appelée lorsqu'un utilisateur sélectionne une catégorie spécifique dans le formulaire. Cette fonction envoie une requête POST à "product_display.php" avec le paramètre "displaySingleCategory" et la valeur de l'ID de catégorie sélectionnée. Ensuite, elle récupère la réponse JSON qui contient les produits correspondants à cette catégorie et les affiche à l'aide de la fonction `rawDisplay()`. Elle appelle également une autre fonction pour afficher les boutons des sous-catégories associées à la catégorie sélectionnée mais elle n'est pas présentée dans la suite de l'extrait.



2. Dans le fichier "product_display.php", une vérification est effectuée pour s'assurer que le paramètre "displaySingleCategory" a été transmis via la requête POST. Ensuite, une instance de la classe "ProductModel" est créée et la méthode `readOnebyForeignKey()` est appelée pour récupérer les produits associés à la catégorie spécifiée. Les résultats sont renvoyés en tant que réponse JSON.

3. Dans le fichier "AbstractModel.php", la méthode `readOnebyForeignKey()` est définie. Elle prend un nom de clé étrangère, une valeur de clé et un paramètre facultatif d'ordre en entrée. Cette méthode construit une requête SQL SELECT pour sélectionner toutes les colonnes de la table spécifiée où la clé étrangère correspond à la valeur fournie. La requête est exécutée et les résultats sont renvoyés sous forme de tableau.

- tableau de bord dynamique

Ces différents morceaux de code interagissent pour récupérer et afficher le nombre total de paniers dans le fichier JavaScript "admin.js" à partir du serveur. À ce stade, il s'agit de permettre à l'administrateur d'avoir une vue globale sur l'activité de son entreprise, en l'occurrence ici, ses ventes.

```
// admin.js (demande côté client du nombre de paniers)
async function fetchCartCount() {
    const response = await fetch("../src/Routes/admin_management.php?countCarts");
    const cartCounts = await response.json();
    // Afficher les résultats
}

// admin_management.php (si demande, instantiation des class)
if (isset($_POST['countCarts'])) {
    $cartModel = new CartModel();
    $totalCarts = $cartModel->countAllCarts();
    echo $totalCarts;
}

// CartModel.php (extends from AbstractModel)
public function countAllCarts(): int
{
    $this->tableName = "carts";
    return $this->countAll();
}

// AbstractModel.php
public function countAll(): int
{
    $requestCountAll = "SELECT COUNT(*) AS total_entries FROM $this->tableName";
    $queryCountAll = self::getPdo()->prepare($requestCountAll);
    $queryCountAll->execute();
    $resultCountAll = $queryCountAll->fetch();
    $totalEntries = $resultCountAll['total_entries'];
    return $totalEntries;
}
```

Lorsque la fonction `fetchCartCount()` est appelée depuis le fichier JavaScript, elle envoie une requête à "admin_management.php" avec le paramètre "countCarts". Le code récupère cette requête et utilise le modèle `CartModel`, qui étend la classe abstraite `AbstractModel`. L'`AbstractModel` fournit une fonction générique `countAll()` qui peut être utilisée pour compter le nombre total d'entrées dans une table spécifique.

Dans le fichier "CartModel.php", la fonction `countAllCarts()` utilise l'héritage pour appeler la fonction `countAll()` de l'`AbstractModel` en spécifiant que la table concernée est "carts" (paniers). Le nombre total de paniers est renvoyé par la fonction `countAllCarts()`.

Ensuite, dans "admin_management.php", le nombre total de paniers est récupéré en appelant la méthode `countAllCarts()` de l'instance de `CartModel`. Ce nombre est renvoyé comme réponse à la requête.

Enfin, dans le fichier JavaScript, la réponse JSON contenant le nombre total de paniers est récupérée et est utilisée pour afficher les résultats comme souhaité. L'AbstractModel agit comme une classe de base générique qui fournit des fonctionnalités communes pour le comptage d'entrées dans une table, tandis que le CartModel spécifique étend cette fonctionnalité pour compter les paniers dans la table "carts".

En d'autres termes, lorsque la fonction `fetchCartCount()` est appelée depuis le JavaScript, elle envoie une requête au fichier PHP "admin_management.php" avec le paramètre "countCarts". Le code PHP récupère cette requête, utilise le modèle CartModel pour compter le nombre total de paniers, puis renvoie ce nombre au fichier JavaScript, où il est utilisé pour l'affichage. L'asynchronicité permet au code de fonctionner même s'il n'a pas encore obtenu les réponses dont il a besoin, et d'y revenir plus tard.

Par ailleurs, il s'agira de noter que, par opposition à un code statique, ce code s'adapte en fonction des réponses du serveur et peut traiter des données variables. Il s'agit donc d'un code dynamique, qui interagit avec des ressources externes, utilise des requêtes HTTP, et utilise des concepts tels que l'héritage et l'abstraction pour créer une structure flexible et modulaire.

The screenshot shows a dashboard interface with a yellow header bar containing navigation links: ACCUEIL, À PROPOS, NOS PRODUITS, and NOS PRODUCTEURS. Below the header is a large yellow section with the word "Dashboard" in a large, bold, white font. At the top of this section, there is a timestamp "Informations relevées le : 17/05/2023 à 11h24" and a green "Actualiser" button. Below the timestamp is a section titled "Chiffres clés" (Key Figures) in bold black text. This section contains two groups of statistics. The first group, enclosed in a red rectangular border, includes "Paniers créés : 7", "Paniers en attente : 5", and "Paniers convertis en ventes : 2". The second group includes "Clients inscrits : 5", "Clients particuliers : 4", and "Clients entreprises : 1".

- sécurité et protection des données grâce aux classes abstraites et héritages : exemple avec la connexion à la base de données

L'exemple le plus représentatif de l'intérêt à utiliser des classes abstraites et la notion d'héritage s'illustre lors de la connexion à la base de données. En effet, cela présente plusieurs avantages :

1. **Réutilisation du code** : En encapsulant la logique de connexion à la base de données dans une classe abstraite, il devient plus facile de réutiliser ce code dans plusieurs classes qui nécessitent une connexion à la base de données. Cela favorise la modularité et évite la duplication de code.
2. **Maintenance simplifiée** : En centralisant la logique de connexion dans une classe abstraite, toute modification ou amélioration apportée à la gestion de la connexion n'a besoin d'être effectuée qu'à un seul endroit. Cela simplifie la maintenance du code et réduit les risques d'erreurs introduites par des modifications dispersées dans différentes classes.
3. **Abstraction des détails de la connexion** : Une classe abstraite permet d'abstraire les détails spécifiques de la connexion à la base de données, tels que le nom de l'hôte, le nom d'utilisateur, le mot de passe, etc. Cela permet de masquer ces informations sensibles et de les centraliser dans une classe dédiée.

Toutefois, la décision de mettre en place un tel procédé a été prise en cours de route et des erreurs de sérialisation, liées à la ré-instanciation d'objets déjà déclarés, se sont présentées. L'utilisation d'une méthode statique a été la solution appropriée pour résoudre ces erreurs. En utilisant une méthode statique pour établir la connexion à la base de données, nous avons contourné la nécessité de sérialiser l'objet de connexion lui-même, car les méthodes statiques sont liées à la classe plutôt qu'à une instance spécifique de la classe. En effet, les méthodes statiques peuvent être appelées directement sur la classe elle-même, sans avoir besoin de créer une instance de la classe. Cela implique que la méthode statique ne peut pas accéder aux propriétés non statiques de la classe et ne peut pas utiliser le mot-clé \$this. Cependant, cela permet d'appeler la méthode directement sur la classe. Cela simplifie l'appel à la méthode de connexion et permet aux autres classes d'obtenir rapidement et facilement une connexion à la base de données.

Le tout, en assurant la sécurité des données, car en effet, si l'exemple qui suit s'attarde sur la connexion à la base de données, il s'agira de noter que toutes les opérations "CRUD" (create, read, update et delete) agissent selon le même fonctionnement.

```
abstract class AbstractModel
{
    protected string $tableName;

    protected ?string $password = null;

    protected static $pdo;

    public function __construct()
    {
    }

    public static function connect()
    {
        $password = (PHP_OS == 'Linux') ? '' : 'root';
        // $pleskPassword = 'eShop123!';
        $dsn = 'mysql:host=localhost;dbname=eShop;charset=utf8';
        // $pleskdsn = 'mysql:host=localhost;dbname=alexandre-aloesode_eShop;charset=utf8';
        self::$pdo = new \PDO($dsn, 'root', $password);
        // self::$pdo = new \PDO($pleskdsn, 'eShop', $pleskPassword);
    }

    protected static function getPdo()
    {
        if (!self::$pdo) {
            self::connect();
        }
        return self::$pdo;
    }
}
```

```
namespace App\Model;

use App\Model\Abstract\AbstractModel;

class UserModel extends AbstractModel
{

    public function __construct()
    {
        parent::connect();
        $this->tableName = 'users';
    }
}
```

- optimisé pour les moteurs de recherche

Le référencement naturel (SEO ou “Search Engine Optimization”) est principalement lié à l’optimisation du contenu d’un site Web et à sa structure technique. Voici un exemple de code HTML optimisé :

<title> : Le titre de la page, il doit être unique, descriptif et contenir des mots-clés pertinents.

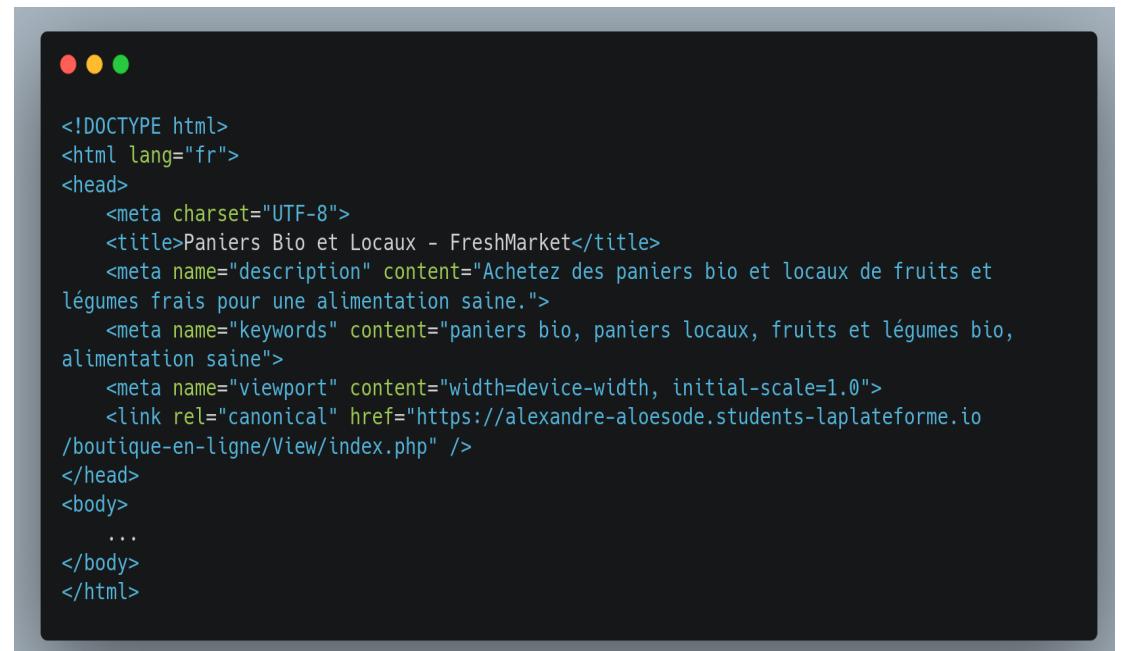
<meta name="description"> : Une description concise de la page utilisée par les moteurs de recherche pour afficher un extrait de la page dans les résultats de recherche.

<meta name="keywords"> : Les mots-clés pertinents pour la page.

<meta name="viewport"> : Indique aux navigateurs comment ajuster la mise en page en fonction de la largeur de l’appareil utilisé.

<link rel="canonical"> : L’URL canonique de la page, utile pour éviter le contenu en double si plusieurs URL mènent à la même page.

Les balises **<header>**, **<nav>**, **<main>**, **<footer>** ne sont pas présentes dans cet extrait mais ont été utilisées pour structurer le contenu de la page.



```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Paniers Bio et Locaux - FreshMarket</title>
    <meta name="description" content="Achetez des paniers bio et locaux de fruits et légumes frais pour une alimentation saine.">
    <meta name="keywords" content="paniers bio, paniers locaux, fruits et légumes bio, alimentation saine">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="canonical" href="https://alexandre-aloesode.students-laplateforme.io/boutique-en-ligne/View/index.php" />
</head>
<body>
    ...
</body>
</html>
```

- évolutivité et maintenabilité grâce à une architecture MVC (Model-View-Controller)

L'architecture MVC assure une séparation claire entre la logique métier et l'affichage du logiciel. La logique métier est gérée dans la couche Modèle, tandis que l'affichage est pris en charge par la couche Vue. Le Contrôleur agit quant à lui en tant qu'intermédiaire entre la vue et le modèle. Il reçoit les entrées de l'utilisateur à travers la Vue, interagit avec le Modèle en conséquence, puis met à jour la Vue pour refléter les changements.

Cette séparation facilite la compréhension et la modification du code, grâce à une meilleure organisation du code, et offre une maintenance simplifiée.



Il est important de noter que bien que le modèle mis en place ne suive pas strictement le modèle MVC traditionnel, les fichiers appelés "routes" dans notre cas ont été introduits pour ajouter une étape de routage, permettant de définir les correspondances entre les URL et les scripts/fonctions à exécuter, s'approchant du concept de routes dans le modèle MVC.

6. Barre de recherche avec auto complétion : une fonctionnalité représentative des compétences à maîtriser

Lorsque l'utilisateur utilise la fonction de recherche avec auto complétion, voici comment les différents fichiers et fonctionnalités s'articulent :

1. “**header.php**” : Ce fichier est inclus dans une autre page et représente la partie d'en-tête contenant le formulaire de recherche. Il contient un formulaire avec un champ de recherche (#field) et une section (#searchResult) où les résultats de recherche seront affichés.
2. “**auto_complete.js**” : Ce fichier contient le code JavaScript qui gère l'interaction utilisateur et la communication avec le serveur pour obtenir les résultats de recherche. Lorsque l'utilisateur saisit des caractères dans le champ de recherche (#field), un événement "keyup" est déclenché. Le code vérifie si le champ de recherche n'est pas vide, puis envoie une requête fetch vers le fichier “search.php” en incluant la valeur du champ de recherche comme paramètre "field". Une fois la réponse reçue, elle est convertie en format JSON. Ensuite, les résultats précédemment affichés sont supprimés, et de nouveaux éléments HTML (balises "a" et "p") sont créés pour afficher les résultats de recherche obtenus. Ces éléments sont ensuite ajoutés à la section de résultat (#searchResult) et stockés dans le tableau suggestionsArray.
3. “**search.php**” : Ce fichier est une route qui est invoquée lorsque le paramètre "field" est présent dans la requête GET. Il initialise une instance de la classe ProductModel et appelle sa méthode *catchProductInfos()* en passant la valeur du paramètre "field" en tant qu'argument. Cette méthode effectue une requête SQL pour récupérer les informations des produits correspondant au mot-clé fourni et renvoie les résultats sous forme de réponse JSON.
4. “**ProductModel.php**” : Ce fichier contient la classe ProductModel qui gère l'accès aux données des produits. La méthode *catchProductInfos(\$word)* de cette classe prend le mot-clé de recherche en argument et exécute une requête SQL pour récupérer les informations des produits dont le nom commence par ce mot-clé. Les résultats sont renvoyés sous forme de réponse JSON.

```
// search.php :  
if (isset($_GET['field'])) {  
    $word = $_GET['field'];  
    $search = new ProductModel;  
    $search->catchProductInfos($word);  
}  
  
// auto_complete.js :  
const field = document.querySelector("#field");  
const searchForm = document.querySelector("#searchForm");  
const suggestionsArray = [];  
const container = document.querySelector("#searchResult");  
  
field.addEventListener("keyup", async (event) => {  
    event.preventDefault();  
    let filledField = field.value;  
  
    if (filledField.length != 0) {  
        fetch("../src/Routes/search.php?field=" + filledField)  
            .then((response) => response.json())  
            .then((response) => {  
                suggestionsArray.forEach((link) => container.removeChild(link));  
                suggestionsArray.length = 0;  
                response.forEach((result) => {  
                    const p = document.createElement("p");  
                    const link = document.createElement("a");  
                    link.setAttribute("href", "singleCard.php?productId=" + result.id);  
                    p.textContent = result.product;  
                    link.appendChild(p);  
                    container.appendChild(link);  
                    suggestionsArray.push(link);  
                });  
            });  
    } else {  
        suggestionsArray.forEach((link) => container.removeChild(link));  
        suggestionsArray.length = 0;  
    }  
});
```

```

// ProductModel.php :
public function catchProductInfos($word)
{
    $req = "SELECT * FROM products WHERE product LIKE '$word%'";
    $catchProduct = self::getPdo()->prepare($req);
    $catchProduct->execute();
    $results = $catchProduct->fetchAll(\PDO::FETCH_ASSOC);

    $json = json_encode($results);
    echo $json;
}

// header.php :
<div id="searchContainer">
    <form action="" class="search-form" id="searchForm">
        <input type="search" name="search" placeholder="Rechercher un produit" id="field">
        <label for="search-box" class="fas fa-search"></label>
    </form>
    <div id="searchResult"></div>
</div>

```

En résumé, lorsque l'utilisateur saisit des caractères dans le champ de recherche (#field), le code JavaScript déclenche une requête vers "search.php", qui utilise la classe ProductModel pour récupérer les résultats de recherche. Les résultats sont ensuite renvoyés au script JavaScript qui les affiche dans la section #searchResult de la page HTML. Ainsi, l'utilisateur peut voir les suggestions de recherche en temps réel pendant qu'il tape, ou efface, dans le champ de recherche.

7. Les vulnérabilités connues et les moyens mis en oeuvre pour y pallier

L'OWASP (ou “Open Web Application Security Project”) est une organisation à but non lucratif qui fournit des ressources et des conseils pour aider les développeurs à renforcer la sécurité de leurs applications web. La sécurité étant l'une des préoccupations majeures de ce projet, voici la manière dont ont été mises en place les bonnes pratiques recommandées par l'OWASP.

a. Contrôle d'accès défaillants :

Ces failles rendent accessibles à des utilisateurs non autorisés des ressources/fonctionnalités sensibles. Cela peut inclure des erreurs de configuration, des priviléges excessifs, des défauts dans la gestion des sessions.

```
<?php if ($_SESSION['user']->getType() == 1) : ?>
<label class="updateLabels" id="updateLabelFirstName" for="updateFirstName">Prénom</label>
<input type="text" name="updateFirstName" class="loginInputs" id="updateFirstName" value=<?= $_SESSION['user']->getFirstName() ?>">

<label class="updateLabels" id="updateLabelLastName" for="updateLastName">Nom</label>
<input type="text" name="updateLastName" class="loginInputs" id="updateLastName" value=<?= $_SESSION['user']->getLastName() ?>">
<?php elseif ($_SESSION['user']->getType() == 2) : ?>
<label class="updateLabels" id="updateLabelCompany" for="updateCompany">Raison Sociale</label>
<input type="text" name="updateCompany" class="loginInputs" id="updateCompany" value=<?= $_SESSION['user']->getCompany() ?>">
<?php endif ?>

<label class="updateLabels" for="updateEmail">Email</label>
<input type="text" name="updateEmail" class="loginInputs" id="updateEmail" value=<?= $_SESSION['user']->getEmail() ?>">
<label class="updateLabels" for="updateAddress">Adresse</label>
<input type="text" name="updateAddress" class="loginInputs" id="updateAddress" value=<?= $_SESSION['user']->getAddress() ?>">
<label class="updateLabels" for="updateZipCode">Code Postal</label>
<input type="number" name="updateZipCode" class="loginInputs" id="updateZipCode" value=<?= $_SESSION['user']->getZipCode() ?>">
<label class="updateLabels" for="updateCity">Ville</label>
<input type="text" name="updateCity" class="loginInputs" id="updateCity" value=<?= $_SESSION['user']->getCity() ?>">
<label class="updateLabels" for="updatePassword">Mot de Passe</label>
<input type="password" name="updatePassword" class="loginInputs" id="updatePassword">
<label class="updateLabels" for="updateConfirmPassword">Confirmez votre mot de passe</label>
<input type="password" name="updateConfirmPassword" class="loginInputs" id="updateConfirmPassword">
<label for="confirmOldPassword">Saisissez votre ancien mot de passe</label>
<input type="password" name="confirmOldPassword" class="loginInputs" id="confirmOldPassword">
```

✓ Il faut tout bloquer par défaut et ne laisser accessible que ce qui est nécessaire : en fonction de son rôle, l'utilisateur peut modifier certains champs et pas d'autres.

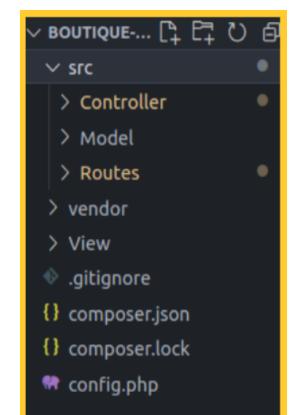
✓ De préférence, un utilisateur ne doit pouvoir modifier que des ressources qui lui appartiennent : il n'a accès qu'aux informations qui le concernent et aucune information sensible n'est en clair.

✓ Il faut pouvoir vérifier que l'utilisateur actuel possède bien les droits qu'il prétend avoir : une confirmation de l'ancien mot de passe est requise même si on est connecté.

✓ Vérifiez que les fichiers de métadonnées ne soient pas exposés : mise en place d'un fichier .gitignore.

✓ Les identifiants de session doivent être invalidés côté serveur pour éviter qu'un pirate ait accès à une session laissée ouverte : après une déconnexion, on détruit la session.

```
if (isset($_POST['disconnect'])) {
    session_destroy();
    header('Location: ../View/index.php');
}
```



b. Défaillances cryptographiques :

Les défaillances cryptographiques font référence à des vulnérabilités dans les mécanismes de cryptographie utilisés pour protéger les données sensibles. Cela peut inclure l'utilisation de clés faibles, d'algorithmes obsolètes, de mauvaises pratiques de gestion des clés, ce qui peut entraîner une compromission de la confidentialité et de l'intégrité des données.

- ✓ Ne stockez que les données dont vous avez besoin et chiffrez les données sensibles exclusivement au repos : le mot de passe n'est pas stocké en session. Aussi, en regard du RGPD, on se doit de justifier la raison pour laquelle on stocke chaque donnée utilisateur.

```
object( App\Model\UserModel )[ 3 ]
protected string 'tableName' => string 'users'    (length=5)
protected ?string 'password' => null
private ?int 'id' => int 46
private ?int 'type' => int 1
private ?string 'firstName' => string 'Alexandre'   (length=9)
private ?string 'lastName' => string 'Aloesode'     (length=8)
private ?string 'company' => *uninitialized*
private ?string 'address' => string '42 rue saint jacques' (length=20)
private ?int 'zipCode' => int 13006
private ?string 'city' => string 'marseille'   (length=9)
private ?string 'avatar' => string 'http://localhost/boutique-en-ligne/View/assets/images/avatars/avatar2.png' (length=73)
private ?string 'verified' => string 'NON'      (length=3)
private ?string 'email' => string 'alexandrealoesode@gmail.com' (length=27)
```

- ✓ Utilisez des méthodes de hachage puissantes (pas MD5 ou SHA1) : nous utilisons bcrypt "PASSWORD_DEFAULT", l'algorithme de hachage par défaut configuré dans PHP. La fonction *password_verify()* est utilisée pour vérifier si un mot de passe correspond à son hachage stocké.

```
● ● ●

$cryptedPassword = password_hash($password, PASSWORD_DEFAULT);

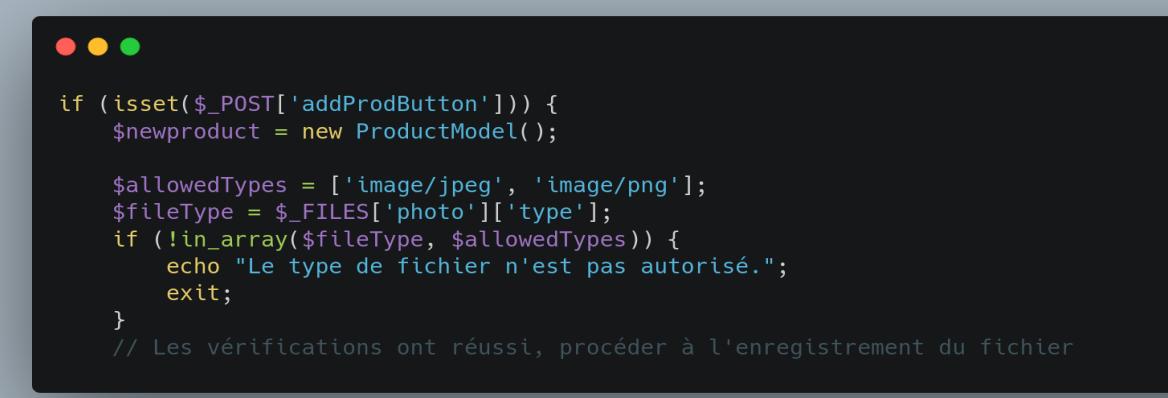
if (password_verify($password, $cryptedPassword)) {
    echo "Mot de passe valide";
} else {
    echo "Mot de passe invalide";
}
```

c. Injections :

Les injections se produisent lorsque des données non fiables sont incorporées dans une commande ou une requête, ce qui peut permettre aux attaquants d'exécuter du code malveillant ou de manipuler les données d'une application de manière non autorisée.

SQL : insertion d'une requête SQL (Insert/Delete/Update) via un input côté client. Un utilisateur malveillant pourrait injecter un script en base de données en l'insérant simplement dans un champ texte qui ne serait pas protégé.

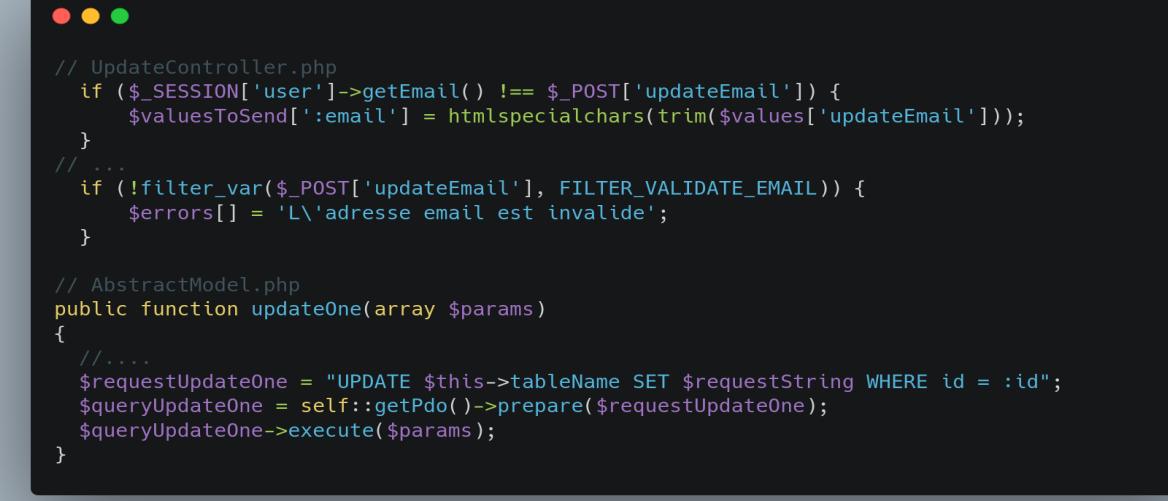
XSS (ou “Cross Site Scripting”) : une des exploitations des XSS les plus utiles pour un pirate est l'injection de code JavaScript dans la page HTML afin de voler le cookie de session de la victime, permettant donc de se connecter à l'application web en se faisant passer pour la victime.



```
if (isset($_POST['addProdButton'])) {
    $newproduct = new ProductModel();

    $allowedTypes = ['image/jpeg', 'image/png'];
    $fileType = $_FILES['photo']['type'];
    if (!in_array($fileType, $allowedTypes)) {
        echo "Le type de fichier n'est pas autorisé.";
        exit;
    }
    // Les vérifications ont réussi, procéder à l'enregistrement du fichier
```

✓ Vérifiez toujours les données qui sont envoyées par l'utilisateur : on vérifie le type de fichier envoyé par l'administrateur au moment de l'ajout de l'image d'un nouveau produit.



```
// UpdateController.php
if ($_SESSION['user']->getEmail() !== $_POST['updateEmail']) {
    $valuesToSend[':email'] = htmlspecialchars(trim($_POST['updateEmail']));
}
// ...
if (!filter_var($_POST['updateEmail'], FILTER_VALIDATE_EMAIL)) {
    $errors[] = 'L\'adresse email est invalide';
}

// AbstractModel.php
public function updateOne(array $params)
{
    //...
    $requestUpdateOne = "UPDATE $this->tableName SET $requestString WHERE id = :id";
    $queryUpdateOne = self::getPdo()->prepare($requestUpdateOne);
    $queryUpdateOne->execute($params);
}
```

✓ Pensez à échapper les données envoyées par l'utilisateur : lorsque l'utilisateur veut mettre à jour son email, on échappe les valeurs saisies dans l'input avec `htmlspecialchars()` et les éventuelles fautes de frappe avec `trim()`, on vérifie qu'il soit un format valide, et on prépare la requête en utilisant PDO (PHP Data Objects, une extension de PHP qui fournit une interface orientée objet pour accéder aux bases de données).

d. Mauvaises configurations de sécurité :

Les mauvaises configurations de sécurité peuvent laisser des vulnérabilités ouvertes, permettant aux attaquants d'exploiter des failles et de compromettre la sécurité du système, des données ou des utilisateurs. Cela arrive en utilisant les configurations par défaut de dépendances que l'on utilise, qui contiennent des failles connues et facilement exploitables.

- Ne jamais utiliser les configurations par défaut de dépendances (ex : root-root comme identifiants de connexion à une base de données)
- Avoir une architecture d'application segmentée qui fournit une séparation entre les composants : la mise en place de l'architecture MVC décrite plus tôt permet de pallier certaines de ces menaces.
- Ne pas donner trop d'indice à un utilisateur dont l'authentification échoue : en cas de saisie d'un mauvais email ou d'un mauvais mot de passe, le message d'erreur est le même et suffisamment vague.



A screenshot of a terminal window with a dark background. In the top-left corner, there are three small colored circles (red, yellow, green). The main area contains the following PHP code:

```
if (!password_verify($_POST['confirmOldPassword'], $savedPassword)) {
    $errors[] = ' Les modifications n\'ont pas été prises en compte.';
}
```

e. Authentification et identification de mauvaise qualité :

Une authentification de mauvaise qualité augmente le risque d'attaques par force brute et de compromission des comptes utilisateurs.

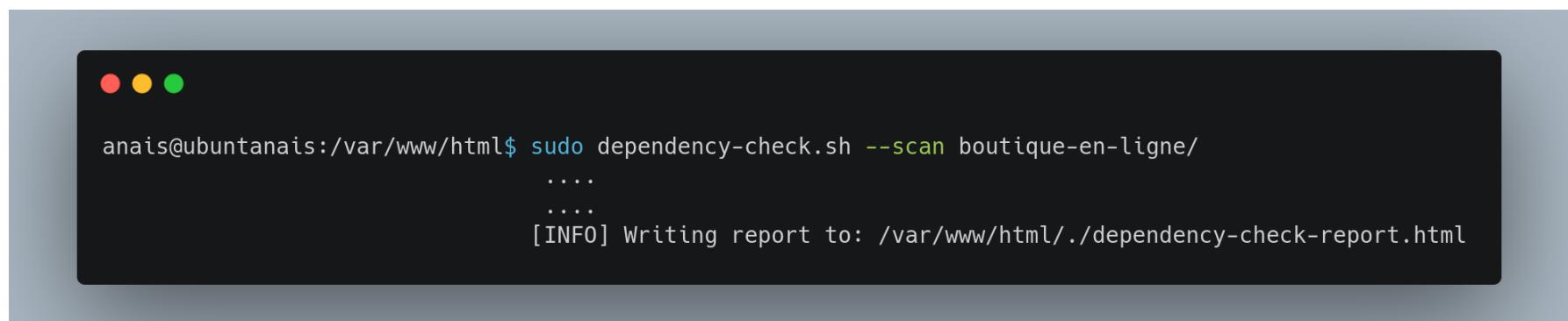
- Faire en sorte que la sécurité des mots de passe soit forte : nous avons interdit les mots de passe de moins de 8 caractères.
- Ne pas donner l'information "mot de passe invalide" qui indique au pirate qu'au moins l'identifiant est bon.

f. Composants vulnérables et obsolètes :

L'utilisation de composants vulnérables et obsolètes expose les applications à des failles de sécurité connues, permettant aux attaquants de les exploiter pour compromettre le système. Pour s'en prémunir il faut faire une veille des composants que l'on utilise.

- Utiliser les dernières versions des dépendances et surveiller les bibliothèques et les composants qui ne sont plus maintenus ou pour lesquels il n'y a plus de correctifs de sécurité.
- OWASP Dependency Check ou OWASP CycloneDX pour vérifier que les composants ne contiennent pas de vulnérabilités connues.

"dependency-check.sh" est un script utilisé pour effectuer une analyse statique des dépendances d'un projet, afin de détecter les vulnérabilités connues dans les bibliothèques utilisées:



```
anaïs@ubuntanais:/var/www/html$ sudo dependency-check.sh --scan boutique-en-ligne/
.....
[INFO] Writing report to: /var/www/html/.dependency-check-report.html
```

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

| Dependency | Vulnerability IDs | Package | Highest Severity | CVE Count | Confidence | Evidence Count |
|-------------------------------------|-------------------|---|------------------|-----------|------------|----------------|
| sweetalert2:11.7.12 | | pkg:npm/sweetalert2@11.7.12 | LOW | 1 | | 8 |

Synk.io est un outil de gestion des dépendances open source qui aide les développeurs à détecter et à résoudre les vulnérabilités dans les bibliothèques de code utilisées dans leurs projets. Il permet d'automatiser l'analyse des dépendances, de recevoir des alertes en cas de failles de sécurité connues, et propose des conseils pour les corriger rapidement. Cela permet aux développeurs de maintenir leurs applications à jour et sécurisées, en réduisant les risques liés aux vulnérabilités des dépendances.

Grâce à cet outil, nous avons pu identifier deux types de menaces - Cross-site Scripting (XSS) et Prototype Pollution :

H Cross-site Scripting (XSS)

SNYK CODE | CWE-79

SCORE 973

```
36     singleCardImage.setAttribute("height", "300px");
37     singleCardDiv.appendChild(singleCardImage);
38
39     const singleCardTitle = document.createElement("p");
40     singleCardTitle.innerHTML = productInfos.name;
```

Unsanitized input from [the document location flows](#) into `innerHTML`, where it is used to dynamically construct the HTML page on client side. This may result in a DOM Based Cross-Site Scripting attack (DOMXSS).

src/Controller/singleCard.js 17 steps in 1 file

M Prototype Pollution

SNYK CODE | CWE-1321

SCORE 556

```
114     singlePageLink.appendChild(cardDescription);
115
116     const cardPrice = document.createElement("p");
117     results[i].price_kg !== null ?
118         cardPrice.innerHTML = (results[i].price_kg/100).toLocaleString("fr-FR", {style:"currency", c
```

Unsanitized input from [data from a remote resource flows](#) into a member access and is used to access a property of [this object](#) by name. This may allow a malicious user to pollute the Object.prototype and cause a crash, remote code execution or logic bypasses.

src/Controller/products.js 14 steps in 1 file

Suite à ce constat, nous nous sommes assurés de valider et filtrer toutes les entrées utilisateur, notamment les commentaires et les notes, côté serveur afin d'empêcher l'injection de scripts malveillants dans le contenu généré dynamiquement. Nous avons également utilisé des fonctions pour échapper les caractères spéciaux avant de les inclure dans le contenu HTML : en utilisant par exemple `textContent` au lieu de `innerHTML` pour afficher du texte brut, ce qui évite l'exécution de scripts indésirables.

L'accès dynamique aux propriétés d'un objet, tel que `results[i].id`, peut présenter des risques de sécurité si le nom est contrôlé par une source externe. Plutôt que d'accéder directement aux propriétés de l'objet par son nom, il serait préférable d'utiliser une méthode sécurisée. Par exemple, comme nous l'avons fait pour les utilisateurs, au lieu d'utiliser `results[i].id`, nous devrions utiliser une méthode qui renvoie la propriété souhaitée, telle que `results[i].getId()`.

g. Manque d'intégrité des données et du logiciel :

Le manque d'intégrité des données et du logiciel peut conduire à des erreurs, à des manipulations non autorisées des données et à une altération du fonctionnement normal du système, compromettant ainsi sa fiabilité et sa sécurité. Les attaquants pourraient potentiellement télécharger leurs propres mises à jour pour les distribuer et les exécuter sur toutes les installations.

- ✓ Mettre en place des vérifications qui permettent de s'assurer que le code qui est disponible est bien celui que vous avez vous même codé (le pipeline CI/CD dispose d'une configuration et d'un contrôle d'accès qui garantit l'intégrité du code passant par les processus d'intégration et de déploiement).
- ✓ Assurez-vous que les dépendances que vous téléchargez sont de confiance : bien que le niveau d'alerte fourni par le script dependency-check.sh ne soit pas alarmant, il nous a semblé plus simple et moins gourmand en ressources d'utiliser le package fourni directement par jsDelivr, un service de distribution de contenu (CDN - Content Delivery Network). Particulièrement, lorsque l'on a noté que le package en question, était le 20ème le plus populaire sur jsDelivr, avec 1 308 118 058 téléchargements durant le mois écoulé.

```
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
```

h. Les axes d'améliorations

Il demeure trois champs d'exploration que nous n'avons pas du tout traités et qu'il s'agira de prendre en compte dans le cadre d'un déploiement réel :

- **Conception non sécurisée (# implémentation) :**

Même une implémentation ultra sécurisée, ne corrigera pas les problèmes qui n'ont pas été gérés en amont.

- ✓ Intégrez les contrôles de sécurité dans les “user stories”
- ✓ Ne demandez pas d'informations personnelles pour confirmer que l'on est bien l'utilisateur que l'on prétend être (social engineering, réseaux sociaux...)

- **Carence des systèmes de contrôle et de journalisation (logs) :**

Sans journalisation, il est impossible de détecter les comportements suspects.

- S'assurer que les enregistrements des journaux sont dans un format standard pour permettre de les intégrer facilement à une solution de gestion de logs centralisée
- La pile Elasticsearch, Logstash, Kibana (ELK) pour ingérer simultanément des données provenant de multiples sources, puis les transformer et les envoyer vers un système de stockage

- **Falsification de requête côté serveur (SSRF) :**

Cette faille permet à un attaquant de forcer un utilisateur authentifié à effectuer des actions sans son consentement via des ports qui auraient dû être inaccessibles.

- Imposer le schéma d'URL, le port et la destination avec une liste positive d'autorisation
- Désactiver les redirections HTTP
- Veiller à la cohérence des URL

8. Description d'une situation ayant nécessité une recherche, à partir d'un site anglophone

Lorsque le site a été déployé sur Plesk, un problème inattendu est survenu dans le panneau d'administration pour la gestion des rôles des utilisateurs alors même que le code était strictement identique. Les différents rôles d'utilisateurs ne s'affichaient pas dans le menu déroulant prévu, ce qui, au-delà d'un problème d'affichage, rendait impossible leur mise à jour.

| EN LOCAL : | SUR PLESK : | | | | | | | | | | | | |
|--|-------------|-----------------|-------|-------|---|-----------------|--|------|----|-------|-------------|---|-----------------|
| <p>Rôles et suppression</p> <table><thead><tr><th>Rôle</th><th>Id</th><th>Email</th></tr></thead><tbody><tr><td>Admin</td><td>1</td><td>admin@admin.com</td></tr></tbody></table> | Rôle | Id | Email | Admin | 1 | admin@admin.com | <p>Rôles et suppression</p> <table><thead><tr><th>Rôle</th><th>Id</th><th>Email</th></tr></thead><tbody><tr><td>Particulier</td><td>1</td><td>admin@admin.com</td></tr></tbody></table> | Rôle | Id | Email | Particulier | 1 | admin@admin.com |
| Rôle | Id | Email | | | | | | | | | | | |
| Admin | 1 | admin@admin.com | | | | | | | | | | | |
| Rôle | Id | Email | | | | | | | | | | | |
| Particulier | 1 | admin@admin.com | | | | | | | | | | | |

À ce stade, la seule piste que nous ayons était de considérer que le passage sur le serveur entraînait un changement de comportement du code. En effet, en analysant le problème à l'aide de console.log, il a été constaté que la valeur du rôle était un nombre en local, mais était considérée comme une chaîne de caractères sur Plesk.

- EN LOCAL :  `number` 11 [admin.js:272:13](#)
- SUR PLESK :  `string` 33 [admin.js:272:13](#)

Des recherches ont révélé que lorsque MariaDB rencontrait une opération ou une situation qui implique à la fois des nombres et des chaînes de caractères, elle effectue une conversion implicite pour s'assurer que les opérations puissent être exécutées de manière cohérente.

Par exemple, si vous avez une colonne de type nombre dans une table MariaDB et que vous utilisez une requête pour récupérer les valeurs de cette colonne, MariaDB effectuera la conversion implicite en convertissant ces nombres en chaînes de caractères, afin de pouvoir renvoyer les résultats sous forme de chaînes.



PRODUCTS SERVICES PRICING RESOURCES ABOUT US DOWNLOAD

[Knowledge Base](#) » [MariaDB Server Documentation](#) » [Using MariaDB Server](#) » [SQL Statements & Structure](#) » [SQL Statements](#) » [Built-in Functions](#) » [String Functions](#) » [Type Conversion](#)

Home

Open Questions

MariaDB Server

MariaDB MaxScale

Type Conversion

Implicit type conversion takes place when MariaDB is using operands of different types, in order to make the operands compatible.

It is best practice not to rely upon implicit conversion; rather use [CAST](#) to explicitly convert types.

Contents

- [1. Rules for Conversion on Comparison](#)
 - [1. Comparison Examples](#)
- [2. Rules for Conversion on Dyadic Arithmetic Operations](#)
 - [1. Arithmetic Examples](#)

↑ String Functions ↑

Regular Expressions Functions

Dynamic Columns Functions

ASCII

BIN

BINARY Operator

[MySQL 8.0 Reference Manual](#) / [Functions and Operators](#) / Type Conversion in Expression Evaluation

version 8.0 ▾

12.3 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts strings to numbers as necessary, and vice versa.

```
mysql> SELECT 1+'1';
-> 2
mysql> SELECT CONCAT(2, ' test');
-> '2 test'
```

Afin de résoudre cette situation, nous devions trouver une solution pour forcer le typage des valeurs des rôles d'utilisateurs :



how to convert a string to a number javascript



La communauté du forum Stackoverflow proposait en premier lieu d'essayer la fonction *Number()*.

The simplest way would be to use the native `Number` function:

```
var x = Number("1000")
```

If that doesn't work for you, then there are the `parseInt`, `unary plus`, `parseFloat with floor`, and `Math.round` methods.

Il s'est donc agi d'ajouter une simple condition dans le code pour contrôler le type de la valeur et la convertir au besoin.

```
if (typeof result[i].type !== "number") {  
    result[i].type = Number(result[i].type);  
}
```